

AMENDMENTS TO THE SPECIFICATION

Please replace the paragraph starting on page 1, line 8 with the following amended paragraph:

This Application is related to the following Application: "Interface Engine Providing A Continuous User Interface," by J. Bret Simister, Adam G. Wolff, Max D. Carlson, Christopher Kimm, and David T. Temkin, U.S. Patent Application No. 10/092,360, filed March 5, 2002, Publication No. 2003/0132959, Attorney Docket No. LZLO-01002US0, filed the same day as the present application and incorporated herein by reference.

Please replace paragraph starting on page 15, line 11 with the following amended paragraph:

The above described mark-up language can be used to describe many different user interfaces. The exact type of user interface is not important to the present invention. One example of a user interface system can be found in the Application: "Interface Engine Providing A Continuous User Interface," by J. Bret Simister, Adam G. Wolff, Max D. Carlson, Christopher Kimm, and David T. Temkin, U.S. Patent Application No. 10/092,360, filed March 5, 2002, Publication No. 2003/0132959, Attorney Docket No. LZLO-01002US0, filed the same day as the present application and incorporated herein by reference.

Please replace paragraph starting on page 25, line 28 with the following amended paragraph:

Figure 5 is a flow chart describing one embodiment of the operation of the presentation server of Figure 4. In step 200, an HTTP request is received at Network Adaptor 160. Protocols other than HTTP can also be used. In step 202, it is determined whether this is an initial request from the particular client. If it is an initial request, then Request Handler 164 associates a session object with the client (step 204). If this is not an initial request, then the already existing session object for that client is accessed and

associated with the request in step 206. The session object stores session specific state information, for example, the state of authentication. In step 208, it is determined whether authentication is required. If this request must be authenticated, then Session Manager 166 calls the Authenticator 168 that is currently registered with the session in step 210. Authenticator 168 attempts to authenticate in step 212. Authenticator 168 marks the status of the request as authenticated or not. In step 214, it is determined whether authentication is successful. If authentication is not successful, then an error message is returned to step 216. In other embodiments, the system may determine not to send an error message or may perform another act in response to a failed authentication. If authentication is successful, the process continues at step 220.

Please replace paragraph starting on page 28, line 26 with the following amended paragraph:

Figure 8 provides a flowchart describing one embodiment for compiling media into byte code. This process can be performed as part of step 376 or step 238. In step 452 a result set is received at the data connection manager from the data source. The result set could be a file or other type of container. The result set includes the media. In step 454, the result set is examined in order to find the media. Any media found in the result set is added to a separate object for each media asset in step 456. The object would be similar to the objects described below with respect to other data. The object would include fields storing attributes such as the name of the media, the format, etc, as well as the actual media itself. In step 458, the actual media is removed from the object and replaced by a reference to the media in step 460. In step 462, the object (including the reference to the media) is compiled to SWF byte code in a similar fashion to other data described below. In step 464, it is determined whether the media data is in an acceptable format. For example, the Flash player described above accepts images in JPEG format. If the media is in a format that is not accepted by the player, then it is transformed using media transformer in step 466 to a format that is accepted by the presentation player. In step 468, the system creates a tag header for the media data. The tag header is in a format suitable for a SWF file, as described above. In step 470, the media data (not including the compiled object) are added to the tag, again in a format suitable for a SWF file. In one embodiment, it is contemplated that one tag will be used to store all of the

media assets. Other embodiments use formats different than SWF and, therefore, would use formats other than SWF tags.

Please replace paragraph starting on page 29, line 17 with the following amended paragraph:

Figure 9 is a flowchart describing a process for compiling view instances. The process of Figure 9 will be performed during step 376. In step 520, the source code for the view instance including the tag name and attributes is accessed. In step 522, a script instruction is created that calls an instantiation view function. This instantiation view function is a function that examines the properties of its parameters and determines what kind of objects to create at run time. The instantiation view function also attaches children to the created objects. In step 524, the tag name and attributes are added to the source description of the object that is passed to the function. In step 526, it is determined whether the view instance includes any children that haven't been processed. If it does, then in step 528, a child including a tag name and attributes are added to the call to the instantiate view function. And the process continues at step 526. If there are no more children to process, then the instruction calling the instantiate view function, which is written in Action Script, is compiled to Action Script byte code in step 540. Compiling Action script to Action Script Byte code is known in the art. In step 542, the byte code is added to an ACTION block. As an example, consider the following code for a view instance:

Please replace paragraph starting on page 33, line 25 with the following amended paragraph:

In step ~~822~~ 824 of Figure 12, the ActionScript is compiled into ActionScript byte code. As described above, compiling action script into action script byte code is known in the art. In one embodiment, step 822 is skipped and the XML code is converted directly into byte code.